EECS 244: Overview of the IC Design Flow

Prof. Kurt Keutzer
EECS
keutzer@eecs.berkeley.edu
No class Monday – Labor Day
The Purpose of the Class Project

- Your first couple years of graduate school are about making the transition from
  - Capable textbook reader $\rightarrow$ scholar of the published research literature
  - Solitary student $\rightarrow$ Active team member
  - Assimilating well defined information $\rightarrow$ pursuing open questions
  - Excellent course/test taker $\rightarrow$ creative researcher
- The project portion of the course is to help you make this transition
Project Outline

- Motivation
- Problem statement
- Prior work
- Investigative approach
- Results
- Summary
- Conclusions
- Future Work
How Conducted

- Research will be conducted in groups of 2-3
- Individual projects highly discouraged
- Research may be coordinated with other class projects: EECS249, EECS290N etc.
- Research will culminate in a:
  - Powerpoint presentation/demo – with explanatory notes
  - Written report
Tips for a Great Project

- Use your skill set
  - Circuits, devices, processing, software development, system-level applications

Great idea:
- Topical – e.g. system level, deep submicron effects, power
  - Tractable – can make an impact in a semester, have all the software, examples, data files that you need

- Get started early
- Get mentorship (senior grad students, post-docs, prof of course)

- Follow deadlines
  - Formulate the problem clearly
  - Formulate your results clearly
Some Successful 244 Projects

- “Constraint Driven Communication Synthesis”, A. Pinto, L. Carloni, A. Sangiovanni-Vincentelli, DAC 2002
Important Class Dates

- Project teams + topics (1 paragraph) due 9/13
- Full project proposals due 9/29
- Exam 1 Handed out 10/6
- Exam 1 collected at BEGINNING of class 10/11
- Preliminary project report due 11/1
- Exam 2 Handed out 11/3
- Exam 2 collected at BEGINNING of class 11/8

- Final project presentations – between 12/6 and 12/8
Application

Motivated by:

- A bright idea
- A market opportunity
  - An emerging market
  - A high growth market
- A technological breakthrough

For example - wireless telephony
Market Opportunity - World’s Cellular Subscribers

Millions

Year


Will provide a ubiquitous infrastructure for wireless data as well as voice

Source: Ericsson Radio Systems, Inc.
**Specification**

- Function, performance (power, delay, area), cost
  - A competitor’s
    - Integrated circuit
    - Data sheet
  - A napkin
  - An industry standard

For example, GSM standard for cellular telephony
System Level Model: GSM System

- System model is organized into major components

![Diagram of GSM System model with major components labeled: RF 900Mhz, IF 70Mhz, 10.7Mhz, SH SAW filter, IF 10.7Mhz, A/D convert, IF 900Mhz, IF 70Mhz, IF 40Ms/sec - 540ks/sec, IF 270.8ks/sec, BB, digital receiver, digital down conv, demand and sync, MLSE Viterbi Eqlz.]

Courtesy Ravi Subramaniam
Mapping onto a system on a chip

- **RAM**
- **µC**
- **DSP CORE**
- **ASIC LOGIC**
- **S/P DMA**

**Phone Book Interface**
- **control protocol**
- **speech quality enhancement**
- **de-intl & decoder**
- **RPE-LTP speech decoder**
- **speech recognition**
- **demodulator and synchronizer**
- **Viterbi equalizer**
Full Wireless Phone Organization
A more complex design – SOC architecture
From specification to design entry

- **Design**: specify and enter the design intent

Verify:
- verify the correctness of design and implementation

Implement:
- refine the design through all phases
Targeting an IC Implementation

- System model is further refined to begin to think about physical implementation

![Diagram showing system model and ASIC logic](image)

HDL Design flow used for ASIC, uC, and DSP

*Courtesy Ravi Subramaniam*
Design Flow

Is the design consistent with the original specification?

Is what I think I want what I really want?
Current Practice: HDL at RTL Level

module foobar (q, clk, s, a, b);
  input clk, s, a, b;
  output q; reg q; reg d;
always @(a or b or s) // mux
  begin
    if (!s )
      d = a;
    else if ( s )
      d = b;
    else
      d = 'bx;
  end // always @(a or b or s)
always @(clk) // latch
  begin
    if ( clk == 1 )
      q = d;
    else if ( clk !== 0 )
      q = 'bx;
  end // always @(clk)
endmodule
An RTL description is always implicitly structural - the registers and their interconnectivity are defined. Thus the clock-to-clock behavior is defined. Only the control logic for the transfers is synthesized.

This approach can be enhanced:
- Register inferencing
- Automating resource allocation

**RTL level**

RTL implies:
\[
\begin{align*}
    a &= b + c; \\
    d &= a + 1; \\
    e &= d \times 2;
\end{align*}
\]

Behavioral implies:
\[
e = 2 \times (b + c + 1)
\]
Verification

- **Design**: specify and enter the design intent

**Verify:**
verify the correctness of design and implementation

**Implement:**
refine the design through all phases
Design Verification

Is the design consistent with the original specification?

Is what I think I want what I really want?
Implementation Verification

Is the implementation consistent with the original design intent?

Is what I implemented what I wanted?
Is the manufactured circuit consistent with the implemented design?

Did they build what I wanted?
Approaches to Design Verification

- Formal verification
  - Model checking - prove properties relative to model
  - Theorem proving - prove properties of a circuit
- Simulation
  - Application of simulation stimulus to model of circuit
- Emulation
  - Implement a version of the circuit on emulator
- Rapid prototyping
  - Create a prototype of actual hardware
Simulation model (HDL)

Simulation driver (vectors)

Simulation monitor (yes/no)
Types of software simulators

- Circuit simulation
  - Spice, Advice, Hspice
  - Timemill + Ace, ADM
- Event-driven gate/RTL/Behavioral simulation
  - Verilog - VCS, NC-Verilog, Turbo-Verilog, Verilog-XL
  - VHDL - VSS, MTI, Leapfrog
- Cycle-based gate/RTL/Behavioral simulation
  - Verilog - Speedsim
  - VHDL – Cyclone
- Generic system-level simulation - SystemC
- Domain-specific simulation
  - SPW, COSSAP,
- Architecture-specific simulation
  - VAST, Axys, Lisatek
Implementation

- **Design**: specify and enter the design intent

Verify:
- verify the correctness of design and implementation

Implement:
- refine the design through all phases
RTL Design Flow

- HDL
- RTL Synthesis
- Module Generators
- Manual Design
- logic optimization
- netlist
- physical design
- layout
- Library

Diagram showing the flow of RTL design process from HDL to physical design, involving module generators, manual design, logic optimization, and netlist creation.
Manual Design

- Performed at
  - Gate level (100 gates/week) /gate-level editor
  - Transistor level (10 - 20 gates week)/tr level editor
- Very expensive in design cost and design time -
- Used for:
  - Analog
  - Leaf cells - libraries, memory cells
  - Datapaths in high performance designs - DSP, microprocessor etc.
Module Generators

- Parameterized generators of actual physical layout
- Typically used for:
  - Memories (word length, #words, # ports)
  - Programmable logic arrays (PLA)
  - Register files
- Occasionally used for:
  - Multipliers
  - General-purpose datapath
  - Datapaths in high performance designs - DSP, microprocessors etc.
Workhorse: RTL Synthesis Flow

Library

RTL Synthesis

HDL

netlist

logic optimization

netlist

physical design

layout
Library

- Contains for each cell:
  - Functional information: cell = a * b * c
  - Timing information: function of
    - input slew
    - intrinsic delay
    - output capacitance
    - non-linear models used in tabular approach
  - Physical footprint (area)
  - Power characteristics
  - Wire-load models - function of
    - Block size
    - Wiring
Reasonable Library Functions

- Inverter, Buffer
- ND2-ND4; NOR2-NOR4; AND2-AND4;
- AOI21 - AOI333; OAI21 - OAI333
- XOR, XNOR
- MUX, Full Adder
- Neg-Edge Triggered D-Flip-Flop
- Pos-Edge Triggered D-FF
- J-K FF
- Above with various clears, enables
- Scan versions of each of the above
- Most of the above in 6 different power sizes:
  - 1x, 2x, 4x, 6x, 8x, 16x
module foobar (q, clk, s, a, b);
input clk, s, a, b;
output q; reg q; reg d;
always @(a or b or s) // mux
begin
  if(~s )
    d = a;
  else if( s )
    d = b;
  else
    d = 'bx;
end // always @(a or b or s)

 translate HDL source into netlist
Logic Optimization

- Perform a variety of transformations and optimizations
  - Structural graph transformations
  - Boolean transformations
  - Mapping into a physical library

pre-optimized

smaller, faster
less power
Optimization Technologies

LOGIC EQUATIONS

TECHNOLOGY-INDEPENDENT OPTIMIZATION

CUBE FACTORING
KERNEL FACTORING

TECH-DEPENDENT OPTIMIZATION (MAPPING, TIMING)

TECHNOLOGY MAPPING
PEEPHOLE OPTIMIZATION

OPTIMIZED LOGIC NETWORK
Kernel and Boolean Factorization

\[ f = a \overline{b} + a \overline{c} + b \overline{a} + b \overline{c} + c \overline{a} + c \overline{b} \]

- Algebraic factorization procedures

\[ f = a(\overline{b} + \overline{c}) + \overline{a}(b + c) + b \overline{c} + c \overline{b} \]

- Boolean factorization produces

\[ f = (a + b + c)(\overline{a} + \overline{b} + \overline{c}) \]

“The decomposition and factorization of Boolean expressions”
RK Brayton, C McMullen, Proc. ISCAS, 1982
Devadas and Keutzer - Chapter 6&7
Optimization Technologies

LOGIC EQUATIONS

TECHNOLOGY-INDEPENDENT OPTIMIZATION

TECH-DEPENDENT OPTIMIZATION (MAPPING, TIMING)

OPTIMIZED LOGIC NETWORK

Cube Factoring
Kernel Factoring
Technology mapping
Peephole optimization
**Reasonable Library**

- Inverter, Buffer
- ND2-ND4; NOR2-NOR4; AND2-AND4;
- AOI21 - AOI333; OAI21 - OAI333
- XOR, XNOR
- MUX, Full Adder
- Neg-Edge Triggered D-Flip-Flop
- Pos-Edge Triggered D-FF
- J-K FF
- Above with various clears, enables
- Scan versions of each of the above
- Most of the above in 6 different power sizes:
  - 1x, 2x, 4x, 6x, 8x, 16x
Input Circuit Netlist

``subject DAG''
Find a Mapping into the Tech Library

Result is a netlist in the technology library

DAGON: Technology Binding and Local Optimization by DAG Matching
Physical Design

- Transform sequential circuit netlist into a physical circuit
  - *place* circuit components
  - *route* wires
  - transform into a mask
- Or for FPGA’s
  - *place* look-up tables
  - *route* wires
Channeled and Channel less

Routing:

- Channel based: Routing only in channels between gates (few metal layers: 2)
- Channel less: Routing over gates (many metal layers: 3 - 6)
- Often split in two steps:
  - Global route: Find a coarse route depending on local density
  - Detailed route: Generate routing layout

J. Christiansen, CERN - EP/MIC
Jorgen.Christiansen@cern.ch
Channeled Gate Array

[Image of a channeled gate array diagram with labels for various components such as terminals, metal layers, power cells, and other structural elements. The diagram includes annotations for VSS, VDD, feedthrough, cell A.11, cell A.14, cell A.132, cell A.23, metal1, metal2, and spacer cells.]
Standard Cell Layout
Channel less Cells

- Arbitrary routing over cells
Physical Design: Overall Flow

Input
- Read Netlist

Partitioning
- Quadratic Placement
- Detailed Placement

Routing Improvement
- Global Routing
- Detailed Routing
- Cost Estimation

Output
- Compaction/finishing
- Write GDSII
Gordian Placement Flow

Fig. 1. Data flow in the placement procedure GORDIAN.
Library, Netlist, and Aspect Ratio

Netlist - >100K cells from library

Size and aspect ratio of core die
Setting up Global Optimization

Fig. 1. Data flow in the placement procedure GORDIAN.
Resulting Layout
Partitioning

Fig. 1. Data flow in the placement procedure GORDIAN.
Layout after Min-cut


A linear-time heuristic for improving network partitions,
Fig. 1. Data flow in the placement procedure GORDIAN.
"Optimal slicing of plane point placements",
van Ginneken, L.P.P. Otten, R.H.J.M.,
Proceedings of the European Design Automation Conference: 12-15 Mar 1990,
322-326.
Other details - slotting constraints

A. E. Dunlop, B. W. Kernighan, 

- Slotting constraints
Generating Final Placement

Fig. 1. Data flow in the placement procedure GORDIAN.
Standard Cell Layout
Physical Design: Overall Flow

- Input
  - Read Netlist

- Placement
  - Quadratic Placement
  - Detailed Placement
  - Partitioning

- Routing
  - Global Routing
  - Detailed Routing
  - Cost Estimation
  - Routing Improvement

- Output
  - Compaction/finishing
  - Write GDSII
Routing

- To simplify routing problem, divide it into two phases
  - Global
  - Detailed

- Global routing
  - Define routing regions
  - Assign nets to regions

- Detailed (Channel) routing
  - Route nets within each region
  - Assign nets to pins
Global Routing

- Grid-Graph Model
- Checker-Board Graph (also use slicing structure)
Channel Routing

- Basic Terminology:
  - Fixed pin positions on top and bottom edges
  - Classical channel: no nets leave channel
  - Three-sided channel possible

A “Greedy” Channel Router, RL Rivest, CM Fiduccia, Design Automation Conference, 1983
**Detail Router 2: Maze Routing**

Basic idea -- wave propagation method (Lee, 1961)

- Breadth-first search
- Back-tracing after finding the shortest path
- Guarantee to find the shortest path

[Diagram of maze routing with labeled nodes and paths]
Physical Design: Overall Flow

Input
- Read Netlist

Placement
- Partitioning
- Quadratic Placement
- Detailed Placement

Routing
- Global Routing
- Detailed Routing
- Cost Estimation
- Routing Improvement

Output
- Compaction/finishing
- Write GDSII
Followed by Compaction

\[ X \text{ then } Y \text{ then } 1D \text{ Compaction} \]

\[ Y \text{ then } X \text{ then } 1D \text{ Compaction} \]
Placed and Routed Standard Cells
Fig. 10. Macrocell design with standard cell blocks $scb8$ and $scb9$. 
Re-visiting Verification

- Design: specify and enter the design intent

Verify:
verify the correctness of design and implementation

Implement:
refine the design through all phases
Implementation Verification

Is the implementation consistent with the original design intent?

Is what I implemented what I wanted?
Use static analysis techniques to verify:

* functionality: • formal equivalence-checking techniques
* and timing: • use static timing analysis
Manufacture Verification (Test)

Is the manufactured circuit consistent with the implemented design?
Did they build what I wanted?

Library/module generators

- HDL
  - RTL Synthesis
    - netlist
      - logic optimization
        - netlist
          - physical design
            - layout

Manual design
Test Synthesis

- Full-chip test requires:

  - JTAG 1149.1
  - Partial scan
  - Full scan
  - Fault simulation for asynchronous interfaces

Diagram of chip components:
- RAM
- S/P DMA
- μC
- ASIC Logic
- DSP Core

BIST
Current Status of RTL Design Flow

- Current RTL design flow is able to produce
  - High speed microprocessors - e.g. Alpha, Pentium Pro > 10M gate-equivalents > 4GHz. (with intervention)
  - System-on-a-chip/Systems on silicon
    - Integration of micro-p, DSP, memory and ASIC on a single die > 10M gate-equivalents >500Hz.
  - Rapid turnaround “Structured” ASIC
    - ISSP-90 HIS from NEC
    - ~ 3M usable gates
    - ~10 Mb SRAM; 10G SerDes
- RTL Design flow now used for FPGA’s as well
  - RTL synthesis provided by independent vendors – e.g. Synplicity – as well as FPGA providers – e.g. Xilinx
  - Place and route provided by FPGA vendors – e.g. Xilinx
How close or far are these 3 solutions
In terms of area, power, programmability ???

**Flexibility**

- Multi-DSP-centered and ASIP-assisted
  - SIMD DSP 1
  - SIMD DSP n
  - ARM
  - Shared Mem
  - Acc 1
  - Acc n
  - Mem
- Reconfigurable Architectures
  - CU 1
  - ...CU n
  - DSP
  - rDPU 1
  - ...rDPU n
  - Mem
- ASIC-centered and DSP-assisted
  - DSP
  - Mem
  - Macro 1
  - ...Macro n

**Software Parts**

- Phy
- L1

**Design Goals**

- Competitive power, area
  - Highest degree of flexibility
  - Simplest Programming Model
  - ≤90nm
- Competitive power, area
  - Flexibility for single family of standards
  - (e.g. WCDMA and CDMA2000)
  - Complex Programming Model
  - 0.18µm – 90nm
- Lowest power consumption
  - No flexibility (e.g. GSM only)
  - 0.5 .. 0.25 µm technology node

Acc: Accelerator
CU: Control Unit
L1: Layer-1 Control
Mem: Memory
PHY: Physical Layer
rDPU: Reconf. Data Processing Unit
SIMD: Single Instruction Multiple Data